CHAPTER V
IMPLEMENTATION AND TESTING

5.1 Implementation

This project has few steps to be done. First, user must launch the application by launching tesGUI.java on terminal.

5.1.1 Compression

5.1.1.1 Select Input File

After launch the application, user can choose the file input. To choose file input, using JfileChooser function as follow

```
private void filenameActionPerformed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event_filenameActionPerformed
    // TODO add your handling code here:
    int returnVal = fileChooser.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION) {
        file = fileChooser.getSelectedFile();
        name = file.getName();
            //System.out.println(name);
        path = file.getAbsolutePath();
        jTextField3.setText(path);
        jLabel4.setIcon(new ImageIcon(path));
}//GEN-LAST:event_filenameActionPerformed
}
```

Figure 5.1.1 JfileChooser Function

5.1.1.2 Compress

After select the file input, there are two options, Compress and Decompress.

For the Compression process, there are a few steps. First, the input image will be read as FileInputStream, convert the image from matrix to vector, count the data frequencies and proceed to build huffman Tree.

```
import java.util.Scanner;
import java.util.*;
import java.io.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import java.util.Scanner;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;



public class HuffmanCode {

  public Tree getHuffmanTree(int[] counts) //input = array frekuensi
  {
    // Buat Priority Queue
    PriorityQueue<Tree> trees = new PriorityQueue<Tree>();
    for (int i = 0; i < counts.length; i++) // cek frekuensi yg tadi
        if (counts[i] > 0)
        trees.offer(new Tree(counts[i], (int)i)); //jika frekuensi > 0, dimasukan ke priority queue

    while (trees.size() > 1) //looping
    {
      // 2 frekuensi trkecil diambil dan dikeluarkan dr queue
      Tree t1 = trees.poll();
      Tree t2 = trees.poll();

      trees.offer(new Tree(t1, t2)); // 2 frekuensi yg td dikeluarkan td ditambahkan trs dimasukan lg ke queue
    }

    return trees.poll();
  }
```

Figure 5.1.2 Get Huffman Tree Code

To build the Huffman Tree, this application use Priority queue from tree. First, initialize the Priority queue, then check the counted frequencies that done before. If the data had frequencies, the data would be inserted into priority queue.

After the insert process done, the next step is build the huffman tree. Take 2 data with smallest frequencies and take it out from priority queue and then add them and insert it again into priority queue. This process would end if the tree size is 1.

After build the Huffman Tree, the code will be generated by traversing from the root to the leaf of huffman tree. As the characteristics of Binary tree, if the nodes have right child, the binary value will be 1, and if it's left, the binary value will be 0. After the code generated, the main class will call out the code to complete the compress process.

```
36  public void assignCode(Node root, String[] codes)
37  { //mbentuk kode huffman
38
39    if (root.left != null) //jika node kiri root ada isinya maka code +0
40    {
41      root.left.code = root.code + "0";
42      assignCode(root.left, codes);
43    }
44    if (root.right != null) //jika node kanan root ada isinya maka code +1
45    {
46      root.right.code = root.code + "1";
47      assignCode(root.right, codes);
48    }
49    else
50    {
51      codes[(int)root.value] = root.code;
52    }
53  }
```

Figure 5.1.3 AssignCode Code

After this process end, proceed to compression process. The data image that converted into vector form processed one by one, substituting the original data value with Huffman code and then write the output file using BitOutputStream method.

```
30        String data="";
31        char chara = 0;
32        for(int i=0;i<text.length();++i)
33        {
34                chara=text.charAt(i);
35                data=data+codes[chara];
36        }
37        count=data.length();
38        //rasio = 100-(count/before)*100;
39        System.out.println("\nJumlah Bit sebelum Huffman= "+before+" bit");
40        System.out.println("Jumlah Bit setelah Huffman= "+count+" bit");
41        //System.out.println("Rasio Kompresi = "+rasio+"%");
42
43        byte [] bytedata;
44        bytedata=data.getBytes();
45        for(int i=0;i<bytedata.length;++i)
46        {
47                if(bytedata[i]==48) bytedata[i]=0; // 48 ASCII 0
48                if(bytedata[i]==49) bytedata[i]=1; // 49 ASCII 1
49                //System.out.println("\nByte = "+bytes);
50        }
51        BitOutputStream bitout = new BitOutputStream(new BufferedOutputStream(new FileOutputStream("compressed file "+file)));
52        try{
53        for (int i = 0; i < bytedata.length; i++)
54        {
55                bitout.write(bytedata[i]);
56        }
57        }finally {
58        bitout.close();
59        }
60        return bytedata.length;
61    }
```

Figure 5.1.4 Huffman Compress Code

5.1.2. Decompression

5.1.2.1 Decompress

For the Decompress process, the input file was the compression result of the image. The Input file reader using BitInputStream because the input file is in the bit form. The process is read the input file bit one by one and traversing huffman tree from the root to the leaf based on the bit file. After reach the leaf, write the values using output stream.

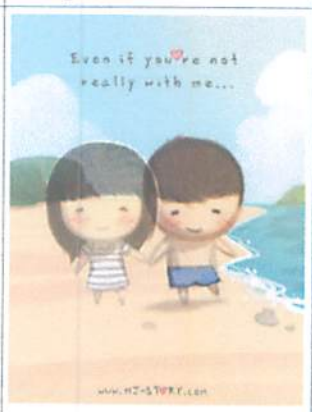
```
17 public class HuffmanDecompress {
18     int w,h,count,z;
19
20     public void decompress (int length, Tree tree, String file) throws FileNotFoundException, IOException{
21         TestGrayscale1 a = new TestGrayscale1();
22         a.readGrayscale(file);
23         w = a.getWidth();
24         h = a.getHeight();
25         int pp = w*h;
26         int n = length;
27         int count =0;
28
29         OutputStream out = new BufferedOutputStream(new FileOutputStream("Hasil Decompress "+file));
30         BitInputStream bip = new BitInputStream(new BufferedInputStream(new FileInputStream("compressed file "+file)));
31         for (int i = 0;i<n ; i++)
32         {
33                 Node nodetree = new Node();
34                 nodetree = tree.root;
35
36                 while (!nodetree.isLeaf())
37                 {
38                   int bit=bip.read();
39                   if (bit==1) nodetree = nodetree.right;
40                   if (bit==0) nodetree = nodetree.left;
41                   count++;
42                 }
43                 out.write(nodetree.value);
44
45         }
46                 out.close();
47                 bip.close();
48         }
49 }
```

Figure 5.1.5 Huffman Decompress code

5.2 Testing

To know how effective huffman coding in image compression, there are the test using this program using various image file as input file. These are the result

Table 5.2.1 Testing Table of the Application

No	Image File	Size	Before (byte)	After (byte)	Ratio
1		450x600	69012	67265	97,47%
2		256x256	15333	15280	99.66%
3		1280x720	103581	103470	99.90%

4		500x667	71324	68966	96.70%
5		600x818	83653	81526	97,46%
6		256x256	20084	20043	99.80%

7		355x357	9838	9615	97.73%
8		236x314	6841	6708	98,06%
9		1280x720	197812	197677	99.94%
10		256x256	10048	10034	99.86%

11		332x300	14481	14283	98,63%
12		464x332	12591	12181	96,74%
13		512x512	38215	37909	99,19%
14		300x400	44303	41956	94,70%

15		500x667	55170	41956	76,04%

5.3 Interface

If user launching tesGUI.java, user can select image as the image input. There are 2 button that can be clicked, Compress and Decompress. Click Compress if user would like to compress the image and Decompress to decompress the compressed files.

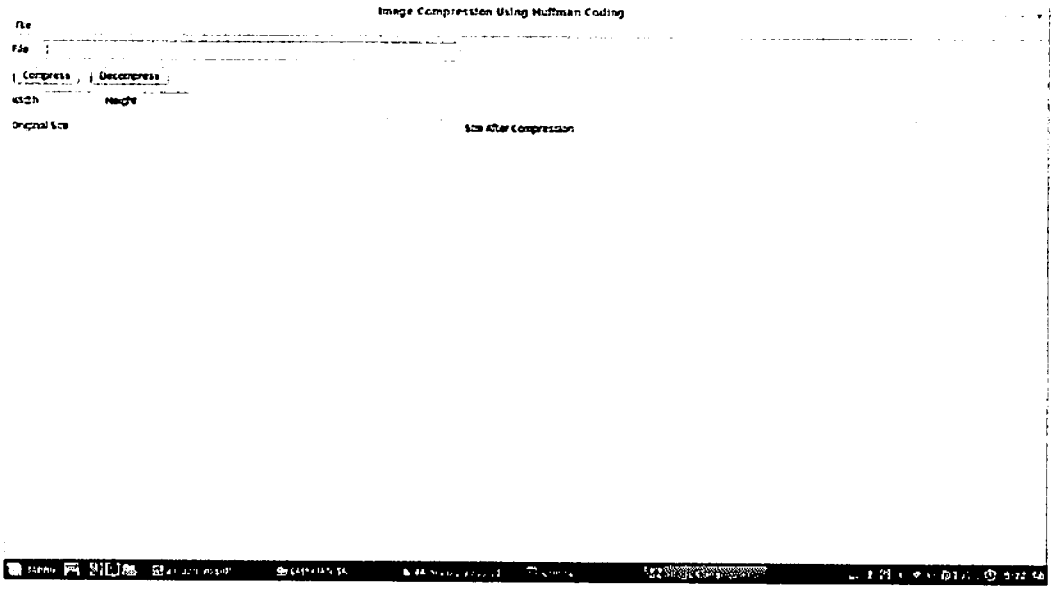


Figure 5.3.1 Main Menu Interface

After selecting the image input, the image displayed in the bottom space. And after compress and decompress process, the result would be displayed on the right side.

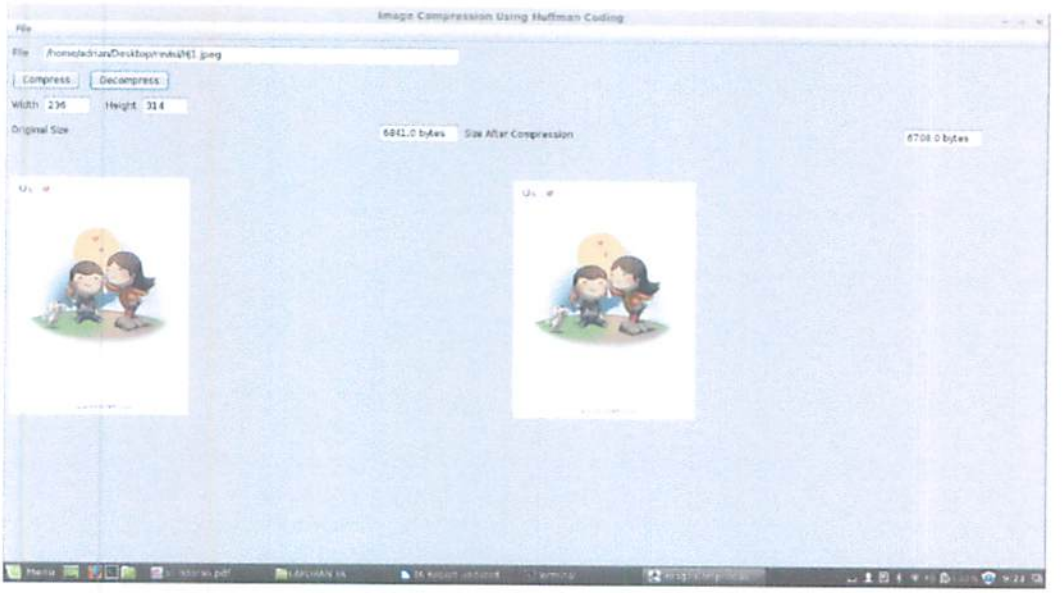


Figure 5.3.2 Result Page